Using pyXS-v2 to process protein solution scattering data

Lin Yang
Revised September, 2012

What is pyXS

pyXS is a set of python scripts that implement functions similar to those available in view.gtk. It offers more flexibility since these codes can be utilized in python code written by users to process the data or control the experiment.

This documentation describes how to use the pyXS package to process solution scattering data. While data collected on the SAXS and WAXS detectors at beamline X9 will be used as examples, the same procedures should also apply for data collected elsewhere, as long as the image files that contain the scattering patterns are readable by the python image library.

Availability and OS Compatibility

The pyXS package can be downloaded from the website of beamline X9 at NSLS: http://www.bnl.gov/ps/x9/software/pyXS.htm . Since pyXS is based on python, in principle it can run under all platforms that support python.

Installation

The pyXS package depends on the following software packages: python, numpy, python image library (PIL), matplotlib. A portion of the code (RQconv) is written in C. Therefore you will need to compile this module if the package from the web link above does not contain a binary suitable for your OS. Doing so will also require the SWIG software.

If you use Linux, you should use the package manager to install these required software packages. If you use Mac, you can install these packages using fink (http://www.finkproject.org/). Alternatively, you may be able to use the native python packages that come OS X (see discussion here: http://www.python.org/download/mac/). But I have not tested this. And the fink works fine for me. If you use Windows, you may have to download these packages individually, and make sure that the python executables are included in your PATH variable. The package from X9 web site includes a WIN32 binary compiled using MinGW.

The web links for the require packages are listed below:

Python: http://www.python.org/download/
Numpy: http://www.scipy.org/Download
Matplotlib: http://matplotlib.sourceforge.net/
PIL: http://www.pythonware.com/products/pil/
(NOTE: An older version of PIL has a bug and cannot read some TIFF files correctly. Make sure you have the most recent version the X9 WAXS data you see do not make sense.)
SWIG: http://www.swig.org/
pyGTK: http://www.pygtk.org/

Overview of pyXS

The pyXS package include the following modules: Data2D.py, RQconv.py, and slnXS.py.

The Data2D module reads (PIL) and displays (matplotlib) 2D scattering patterns and perform the conversion between pixel position and receiprocal space coordinates, q or (q_r, q_z) (see more descriptions of X-ray scattering in the documentation of view.gtk). Data2D relies on C code RQconv.c to convert data, based on scattering geometry defined by the structure ExpPara. The line profile extraction and annotation functions of view.gtk can be similarly accomplished using functions defined in this module.

The slnXS module is used to process 1D solution scattering data produced by Data2D. It can average from multiple scattering patterns, perform background subtraction (dark current and buffer scattering) based on transmitted beam intensity, and combine the SAXS and WAXS data simutaneously collected on the two detectors at X9. This module also provides rudimentary capabilities for producing Gunier plots and for approximate p(r) function calculations.

There are significant differences the between the original version of pyXS and pyXS-v2 in the way they process solution scattering data. In the original version, multiple images collected from the same sample are first converted into 1D data and subtracted for detector dark current; these 1D data are then averaged together and subtracted for buffer scattering. These steps are performs separately for the SAXS and WAXS images. The 1D SAXS and WAXS data after buffer subtraction are then merged into the final data that span the entire q-range. In pyXS-v2, merging of SAXS and WAXS data occurs immediately after dark current subtraction. The merged SAXS/WAXS 1D data are then averaged together and subtracted for buffer scattering. PyXS-v2 also offers the option of buffer scattering subtraction based on high-q data (water scattering).

Setting up for data processing

For data conversion to work properly, the parameters that describe the scattering geometry must be defined. At X9, these parameters are usually defined in a file named exp_setup.py, produced by the beamline staff before user experiments. You can simply import it into your python codes that utlize pyXS. An example of the exp_setup.py file is shown in the appendix.

The scattering geometry is established for the SAXS and WAXS detectors in exp_setup.py. If only one detector is used, only one ExpPara needs to be defined. In order to define the parameters in ExpPara, the scattering pattern from a standard powder sample is measured. The simulated powder rings based on these parameters are then compared to the measured standard pattern. The parameter are adjusted to best match the two. This process can be done interactively using view.gtk (see the documentation of view.gtk). An automatic refinement routine is planned but not yet implemented.

```
From exp_setup.py:

ew = ExpPara()                 # create a new ExpPara
ew.wavelength = 0.874          # X-ray wavelength
ew.bm_ctr_x = 35               # pixel position of the X-ray beam
ew.bm_ctr_y = 1035             #
ew.ratioDw = 4.58              # this is the sample-to-detector distance divided by
                               #      the width of the detector
ew.det_orient = 45             # orientation of the detector
ew.det_tilt = -19              #      0, 0, 0 for the SAXS detector
ew.det_phi = 0                 #
ew.grazing_incident = False    # always false for solution scattering
ew.flip = True                 # True if the 2D image must be flipped diagonally
                               #   this is the case for the WAXS data collected at X9
ew.incident_angle = 0.2        # only meaningful for grazing incident scattering
ew.sample_normal = 0           # only meaningful for grazing incident scattering
```

Next the *q*-grid for the 1D SAXS and WAXS scattering profile are defined. Note that the grid does not have evenly spaced data points. However, a non-uniform data grid (consisting of sections of uniform grids) must be pre-processed by mod_grid() to ensure that the azimuthally averaged data do not contain artifacts.

```
From exp_setup.py:

qgrid = mod_qgrid( np.hstack((np.arange(0.004,0.05,0.001),
                   np.arange(0.05,0.1,0.002),
                   np.arange(0.1,0.5,0.005),
                   np.arange(0.5,1.0,0.01),
                   np.arange(1.0,2.05,0.03))))
```

The masks are defined to block off parts of the scattering pattern that should not be included in the conversion from the 2D scattering patterns to 1D scattering profiles. The mask files are specified when loading the dark current data, which are saved in pickled files and are not recreated during data processing if they already exist. Note that the exposure time for the dark current data should equal that for the actual data.

```
From exp_setup.py:

# this file correspond to the averaged dark current
# change/rebuild this file for new exposure time or qsaxs/qwaxs
DARK_FILE_S="dark-s.pkl"
DARK_FILE_W="dark-w.pkl"

if os.path.isfile(DARK_FILE_S) and os.path.isfile(DARK_FILE_W):
    pkl_file = open(DARK_FILE_S, 'rb')
    sdark = pk.load(pkl_file)
    pkl_file.close()
    pkl_file = open(DARK_FILE_W, 'rb')
    wdark = pk.load(pkl_file)
    pkl_file.close()
else:
    sdark = Data1d()
    wdark = Data1d()
    sdark.load_dark_from_2D(["dark1a.90s_SAXS",
                             "dark1b.90s_SAXS",
                             "dark1c.90s_SAXS",
                             "dark1d.90s_SAXS",
                             "dark1e.90s_SAXS"],
                            es,"mask.SAXS",qgrid)
    wdark.load_dark_from_2D(["dark1a.90s_WAXS",
                             "dark1b.90s_WAXS",
                             "dark1c.90s_WAXS",
                             "dark1d.90s_WAXS",
                             "dark1e.90s_WAXS"],
                            ew,"mask.WAXS",qgrid)
    # pickle doesn't like PIL objects
    sdark.exp_para = None
    wdark.exp_para = None
    pkl_file = open(DARK_FILE_S, 'wb')
    pk.dump(sdark, pkl_file)
    pkl_file.close()
    pkl_file = open(DARK_FILE_W, 'wb')
    pk.dump(wdark, pkl_file)
    pkl_file.close()
```

The format of the mask file is similar to that used in view.gtk, except that polygons can also be used. Here is an example of the mask file (used for the WAXS data collected at X9):

```
From mask.WAXS:

# A mask file should contain geometric shapes, each defined on one line.
# "h" defines a "hole", i.e. inverse of the circle, with pixels outside of the cicle
# blocked off. The parameters are the center (x,y) and the radius.
h       518     518     645
# "p" defines a polygon. The parameters define the vertices (x,y) of the polygon.
p       0       0       0       250     250     0       0       0
p       750     0       1042    0       1042    292     750     0
p       750     1042    1042    1042    1042    750     750     1042
# "r" defines a rectangle. The parameters are the center (x,y), width and height, and the
# rotation of the rectangle about its center.
r       536     521     4       1042    0
r       197     521     1       1042    0
r       518     521     1       1042    0
r       791     521     1       1042    0
```

The flat field data are loaded and saved similarly. For the data collected X9, flat field correction for WAXS data is usually necessary.

```
From exp_setup.py:

# flat field correction includes the incident angle correction
FLAT_FILE_S="flat-s.pkl"
FLAT_FILE_W="flat-w.pkl"

if os.path.isfile(FLAT_FILE_W):
    pkl_file = open(FLAT_FILE_W, 'rb')
    wflat = pk.load(pkl_file)
    pkl_file.close()
else:
    fdark = Data1d()
    wflat = Data1d()
    fdark.load_dark_from_2D(["Feb09-dark-00.300s_WAXS",
                             "Feb09-dark-01.300s_WAXS",
                             "Feb09-dark-02.300s_WAXS",
                             "Feb09-dark-03.300s_WAXS",
                             "Feb09-dark-04.300s_WAXS",
                             "Feb09-dark-05.300s_WAXS"],
                            ew,"mask.WAXS",wdark.qgrid)
    wflat.load_dark_from_2D(["Feb09-bright-00.300s_WAXS",
                             "Feb09-bright-01.300s_WAXS"],
                            ew,"mask.WAXS",wdark.qgrid)
    wflat.data -= fdark.data
    wflat.d2data -= fdark.d2data
    wflat.err += fdark.err
    wflat.save("wflat.dat")
    wflat.exp_para = None
    del wflat.mask
    pkl_file = open(FLAT_FILE_W, 'wb')
    pk.dump(wflat, pkl_file)
    pkl_file.close()
```

In the event that you suspect that the exp_setup.py file provided by the beamline staff is incorrect, you can compare the simulated powder rings to the standard patterns using view.gtk or the disp.py script in the appendix of this document and revise the parameters or the mask accordingly.

Data processing

Data processing is accomplished in the proc_SWAXS() and proc_SAXS() functions defined in the slnXS module. The arguments of these functions are commented below:

```
From proc.py:

d1 = proc_SWAXS(sys.argv[1].split(),        # sample files
                sys.argv[2].split(),        # buffer files
                sdark,wdark,                # dark current data defined in exp_setup
                qmax=0.17,qmin=0.125,       # q-range for merging SAXS/WAXS data
                reft=-1,                    # if >0, reference trans value
                conc=float(sys.argv[3]),    # protein solution concentration
                save1d=True,                # save the ave files
                waxsflat=wflat,             # flat field data defined in exp_setup
                fix_scale=-36.95            # if >0, scaling factor between SAXS and WAXS
                )
```

The proc_SWAXS() and proc_SAXS() functions are defined as the following:

```
From slnXS.py:

def proc_SWAXS(sfns,bfns,sdark,wdark=None,qmax=-1,qmin=-1,reft=-1,save1d=False,conc=0.,
               saxsflat=None,waxsflat=None,fix_scale=-1):
    ds = avg_SWAXS(sfns,sdark,wdark,qmax,qmin,reft,plot_data=True,save1d=save1d,
                   saxsflat=saxsflat,waxsflat=waxsflat,fix_scale=fix_scale)
    db = avg_SWAXS(bfns,sdark,wdark,qmax,qmin,reft,plot_data=True,save1d=save1d,
                   saxsflat=saxsflat,waxsflat=waxsflat,fix_scale=fix_scale)
    vfrac = 0.001*conc/PROTEIN_WATER_DENSITY_RATIO
    ds.bkg_cor(db,1.0-vfrac,plot_data=True)
    return ds


def proc_SAXS(sfns,bfns,sdark,reft=-1,save1d=False,conc=0.,saxsflat=None):
    ds = avg_SWAXS(sfns,sdark,reft=reft,plot_data=True,save1d=save1d,saxsflat=saxsflat)
    db = avg_SWAXS(bfns,sdark,reft=reft,plot_data=True,save1d=save1d,saxsflat=saxsflat)
    vfrac = 0.001*conc/PROTEIN_WATER_DENSITY_RATIO
    ds.bkg_cor(db,1.0-vfrac,plot_data=True)
    return ds
```

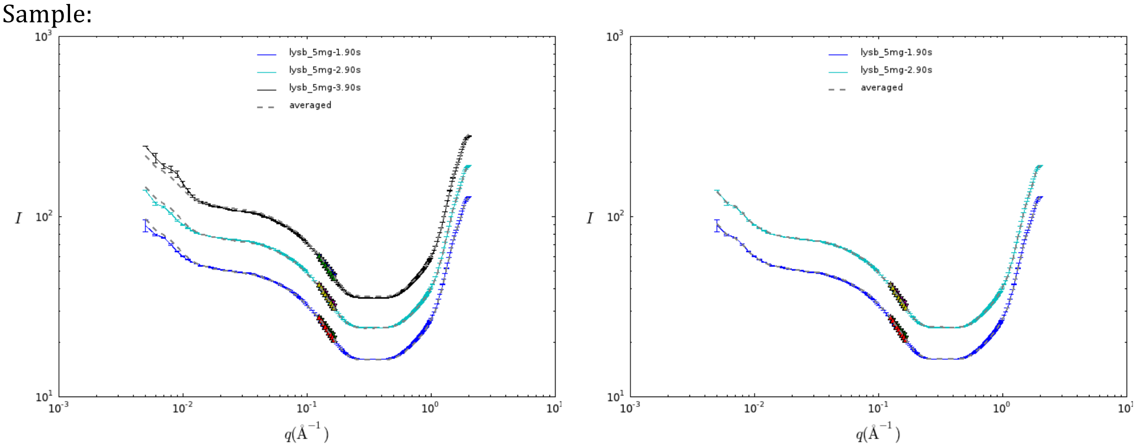
Both functions first convert the 2D images into 1D data by calling avg_SWAXS(), which processes the 2D image files with _SAXS and _WAXS suffixes depending on whether the dark current data are provided. Buffer scattering is then subtracted based on the protein concentration, which should be considered as only a simple scaling factor.

Note that the data and scripts shown below are contained in the Example.Sol directory included in the software package.

1. Conversion of 2D data into 1D scattering profiles

Conversions of the 2D images into 1D scattering profiles take place within avg_SWAXS(). This function produces two plots, showing the merged sample and buffer data converted from each set of SAXS/WAXS pairs, and the average of all sets for the given sample/buffer. Based on these plots, you should re-run the proc.py script with the outlier image omitted.

Sample:



Buffer:

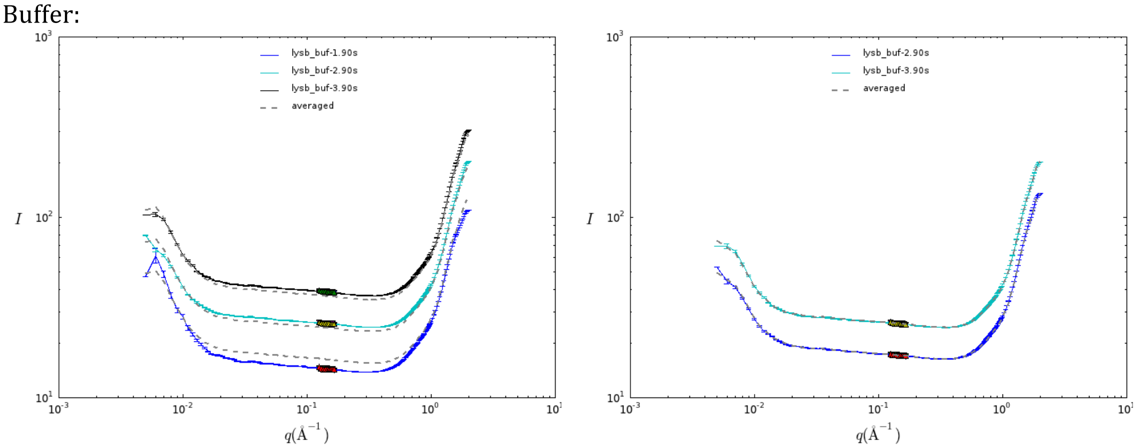


Fig.1 Plots generated by avg_SWAXS(). It is clear from the plots on the left that one of the scattering profiles is significantly different from the other two. This set of data is therefore discarded. The reason for the difference is likely streaking around the beamstop.

You should also pay close attention to how well the SAXS and WAXS data match within the overlapping *q*-range. Flat field correction for the WAXS data is usually necessary. When necessary, adjust the parameters qmax and qmin in proc.py to pick the *q*-range within which the SAXS and WAXS data are scaled to match intensity. If the mismatch is significant, you should also make sure that the scattering geometry defined in exp_setup is correct.

2. Buffer scattering subtraction

Subtraction of buffer scattering is done within proc_SWAXS() or proc_SAXS() as well. Again, the protein concentration can be used as a scaling factor to adjust the high-q intensity in the final data.

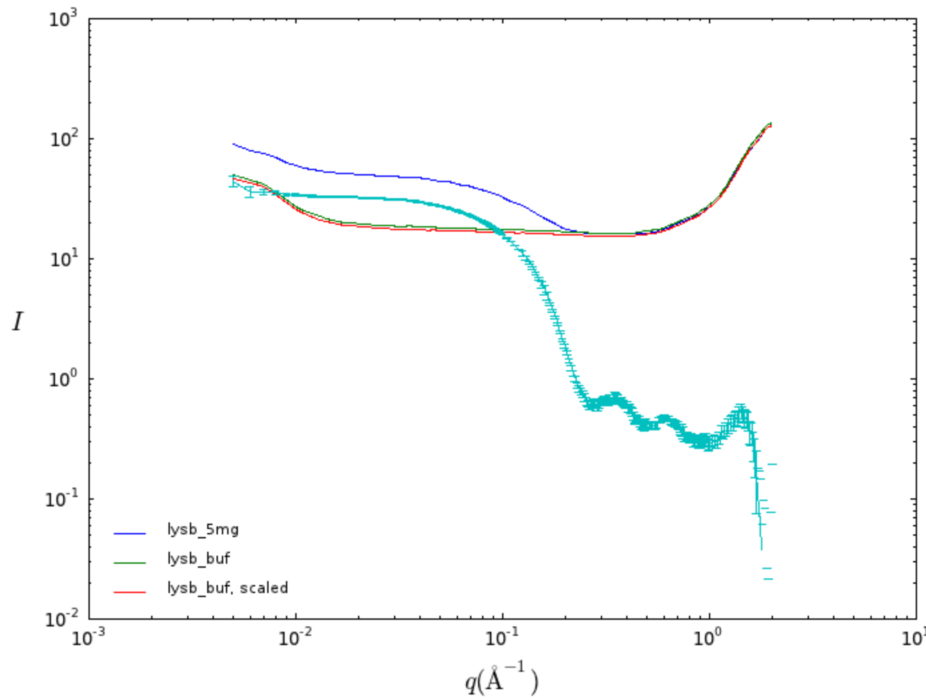


Fig.2 A plot generated by proc_SWAXS(). The sample scattering (red), buffer scattering (red), scaled buffer scattering (green) and the result of background-subtracted data (cyan) are shown.

3. Gunier fit

A Gunier plot can be generated using Data1d.plot_Guinier(). The Guinier fit is performed within the specified q-range. If fix_qe=False, pyXS will adjust the ending point of the q-range until it is below $1/R_g$. A call to Data1d.plot_Gunier() returns the best-fit I_0 and R_g values. Data1d.plot_Guinier() is called from the analyze() function in proc.py.

```
From proc.py:

analyze(d1,qstart=0.02,qend=0.08,fix_qe=False, qcutoff=0.9,dmax=100)
```

```
From slnXS.py:


def analyze(d1,qstart,qend,fix_qe,qcutoff,dmax):
    plt.figure(figsize=(14,5.5))
    plt.subplot(121)
    (I0, Rg) = d1.plot_Guinier(qs=qstart, qe=qend, fix_qe=fix_qe)
    print "I0=%f, Rg=%f" % (I0, Rg)

    plt.subplot(122)
    d1.plot_pr(I0,Rg,qmax=1.2,dmax=100.)

    plt.subplots_adjust(bottom=0.15,wspace=0.25)
```

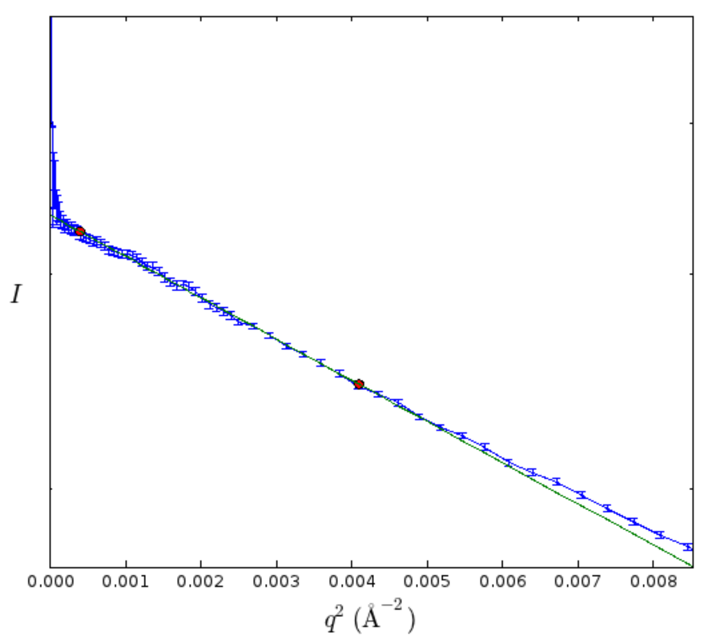


Fig.3 The plot generated by the analyze() function defined in slnXS.py. The fit is performed within the range specified by the qstart and qend arguments, shown as the red dots.

Appendix: example codes

1. exp_setup.py

```
import numpy as np
import cPickle as pk

PYXS_PATH='/Users/lyang/pro/pyXS-v2'
import sys, os
PYXS_PATH in sys.path or sys.path.append(PYXS_PATH)

from Data2D import *
from RQconv import *
from slnXS import *
import matplotlib.pyplot as plt

es = ExpPara()
es.wavelength = 0.874
es.bm_ctr_x = 444
es.bm_ctr_y = 311
es.ratioDw = 40.8
es.det_orient = 0
es.det_tilt = 0
es.det_phi = 0
es.grazing_incident = False
es.flip = False
es.incident_angle = 0.2
es.sample_normal = 0

ew = ExpPara()
ew.wavelength = 0.874
ew.bm_ctr_x = 35
ew.bm_ctr_y = 1035
ew.ratioDw = 4.58
ew.det_orient = 45
ew.det_tilt = -19
ew.det_phi = 0
ew.grazing_incident = False
ew.flip = True
ew.incident_angle = 0.2
ew.sample_normal = 0

# the q grid can be defined arbitrarily
#qgrid = np.arange(0.01,2.05,0.01)
qgrid = mod_qgrid( np.hstack((np.arange(0.004,0.05,0.001),
                   np.arange(0.05,0.1,0.002),
                   np.arange(0.1,0.5,0.005),
                   np.arange(0.5,1.0,0.01),
                   np.arange(1.0,2.05,0.03))))

# this file correspond to the averaged dark current
# change/rebuild this file for new exposure time or qsaxs/qwaxs
DARK_FILE_S="dark-s.pkl"
DARK_FILE_W="dark-w.pkl"

if os.path.isfile(DARK_FILE_S) and os.path.isfile(DARK_FILE_W):
    pkl_file = open(DARK_FILE_S, 'rb')
    sdark = pk.load(pkl_file)
    pkl_file.close()
    pkl_file = open(DARK_FILE_W, 'rb')
    wdark = pk.load(pkl_file)
    pkl_file.close()
else:
    sdark = Data1d()
    wdark = Data1d()
    sdark.load_dark_from_2D(["dark1a.90s_SAXS",
                             "dark1b.90s_SAXS",
                             "dark1c.90s_SAXS",
```

```
                             "dark1d.90s_SAXS",
                             "dark1e.90s_SAXS"],
                            es,"mask.SAXS",qgrid)
    wdark.load_dark_from_2D(["dark1a.90s_WAXS",
                             "dark1b.90s_WAXS",
                             "dark1c.90s_WAXS",
                             "dark1d.90s_WAXS",
                             "dark1e.90s_WAXS"],
                            ew,"mask.WAXS",qgrid)
    # pickle doesn't like PIL objects
    sdark.exp_para = None
    wdark.exp_para = None
    pkl_file = open(DARK_FILE_S, 'wb')
    pk.dump(sdark, pkl_file)
    pkl_file.close()
    pkl_file = open(DARK_FILE_W, 'wb')
    pk.dump(wdark, pkl_file)
    pkl_file.close()

sdark.exp_para = es
wdark.exp_para = ew

# flat field correction includes the incident angle correction
FLAT_FILE_S="flat-s.pkl"
FLAT_FILE_W="flat-w.pkl"

if os.path.isfile(FLAT_FILE_W):
    pkl_file = open(FLAT_FILE_W, 'rb')
    wflat = pk.load(pkl_file)
    pkl_file.close()
else:
    fdark = Data1d()
    wflat = Data1d()
    fdark.load_dark_from_2D(["Feb09-dark-00.300s_WAXS",
                             "Feb09-dark-01.300s_WAXS",
                             "Feb09-dark-02.300s_WAXS",
                             "Feb09-dark-03.300s_WAXS",
                             "Feb09-dark-04.300s_WAXS",
                             "Feb09-dark-05.300s_WAXS"],
                            ew,"mask.WAXS",wdark.qgrid)
    wflat.load_dark_from_2D(["Feb09-bright-00.300s_WAXS",
                             "Feb09-bright-01.300s_WAXS"],
                            ew,"mask.WAXS",wdark.qgrid)
    wflat.data -= fdark.data
    wflat.d2data -= fdark.d2data
    wflat.err += fdark.err
    wflat.save("wflat.dat")
    wflat.exp_para = None
    del wflat.mask
    pkl_file = open(FLAT_FILE_W, 'wb')
    pk.dump(wflat, pkl_file)
    pkl_file.close()
```

2. disp.py

```
#!/sw/bin/python

from exp_setup import *
import matplotlib as mpl
import sys

if (len(sys.argv) < 2):
    print "Usage: disp.py file_name_root"
    exit()
else:

    saxs_current_file = sys.argv[1]+'_SAXS'
    waxs_current_file = sys.argv[1]+'_WAXS'
```

```
    if os.path.isfile(saxs_current_file) and os.path.isfile(waxs_current_file):
        plt.figure(figsize=(10,6))
    else:
        plt.figure(figsize=(6,6))

    if os.path.isfile(saxs_current_file):
        if os.path.isfile(waxs_current_file):
            plt.subplot(121)
        ax1 = plt.gca()
        dsaxs = Data2d(saxs_current_file)
        dsaxs.set_exp_para(es)
        pax1 = Axes2dplot(ax1,dsaxs)
        pax1.plot(mask=sdark.mask)

        if (len(sys.argv)>2):
            pax1.add_dec("Q 0.1076 72 r-")
            pax1.add_dec("Q 0.2152 72 r-")
            pax1.add_dec("Q 0.3228 72 r-")
            pax1.add_dec("Q 0.4304 72 r-")

    if os.path.isfile(waxs_current_file):
        if os.path.isfile(saxs_current_file):
            plt.subplot(122)
        ax2 = plt.gca()
        dwaxs = Data2d(waxs_current_file)
        dwaxs.set_exp_para(ew)
        pax2 = Axes2dplot(ax2,dwaxs)
        pax2.plot(mask=wdark.mask)

        if (len(sys.argv)>2):
            pax2.add_dec("Q 0.1076 72 r-")
            pax2.add_dec("Q 0.2152 72 r-")
            pax2.add_dec("Q 0.3228 72 r-")
            pax2.add_dec("Q 0.4304 72 r-")
            pax2.add_dec("Q 0.5380 72 r-")
            pax2.add_dec("Q 0.6456 72 r-")
            pax2.add_dec("Q 0.7532 72 r-")
            pax2.add_dec("Q 0.8608 72 r-")
            pax2.add_dec("Q 0.9684 72 r-")
            pax2.add_dec("Q 1.0760 72 r-")
            pax2.add_dec("Q 1.37 72 r-")

    plt.show()
```

3. proc.py

```
#!/sw/bin/python
from exp_setup import *

import sys
import slnXS
slnXS.trans_mode = slnXS.TRANS_FROM_WAXS
slnXS.WAXS_THRESH = 100

if len(sys.argv)<3:
    print "Usage: proc.py sample buf protein_conc out_put_file"
    print "multiple files can be given for sample and buf"
    print "e.g. ./proc.py \"lys20 file2\" lysbuf4 3.7 lyso20.dat"
    exit()
else:
    # set plot_data=True to see curves from the individual files
    # a label can be given to the averaged curve
    d1 = proc_SWAXS(sys.argv[1].split(),
                    sys.argv[2].split(),
                    sdark,wdark,
                    qmax=0.17,qmin=0.125,reft=-1,
                    conc=float(sys.argv[3]),
                    save1d=True,
```

```
                     waxsflat=wflat,
                     fix_scale=-36.95
                     )
    d1.save(sys.argv[4])

    analyze(d1,qstart=0.02,qend=0.08,fix_qe=False,qcutoff=0.9,dmax=100)

    plt.show()
```

4. proc-SAXS.py

```
#!/sw/bin/python2.6
from exp_setup import *

import sys
import slnXS
# this assumes that the SAXS data does not contain scattering intensity that can
# be used as a reference for scaling
slnXS.trans_mode = slnXS.TRANS_FROM_BEAM_CENTER

if len(sys.argv)<3:
    print "Usage: proc.py sample buf protein_conc out_put_file"
    print "multiple files can be given for sample and buf"
    print "e.g. ./proc.py \"lys20 file2\" lysbuf4 3.7 lyso20.dat"
    exit()
else:
    # set plot_data=True to see curves from the individual files
    # a label can be given to the averaged curve
    d1 = proc_SAXS(sys.argv[1].split(),
                   sys.argv[2].split(),
                   sdark,reft=-1,
                   conc=float(sys.argv[3]))
    d1.save(sys.argv[4])

    analyze(d1,qstart=0.02,qend=0.08,fix_qe=False,qcutoff=0.9,dmax=100)

    plt.show()
```